

FHWS

iOS Programmierung (mit Swift)

Peter Braun, Florian Bachmann & Andreas Wittmann
[@pe_braun](https://twitter.com/pe_braun) [@florianbachmann](https://twitter.com/florianbachmann) [@anwittmann](https://twitter.com/anwittmann)

Deutsche Telekom AG
FHWS - Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt
#FHWSSwift

Agenda

1. **Introduction** – Organisatorisches
2. **First iOS-Project** – Hello World, **First iOS-Project** – Still Hello World (now with Code 😊)
3. **Swift**, Wait!, What about Objective-C?, Why Swift?
4. **A (not so) Quick Tour**
5. **Documentation**
6. **The basics** – iOS Architecture & more
7. **User Interfaces** – View Controller, Auto Layout & Size Classes
8. **Storyboard & Segues**
9. **Tables & NavigationController**
10. **TabBarController**
11. **Notifications**
12. **PickerViews**
13. **Touches, Gestures, 3D Touch, Peek & Pop**
14. **ScrollView & StackViews**
15. **Networking** – JSON & Dependency Managers
16. **WebKit**
17. **Maps**
18. **Storage & Data persistency** – NSUserDefaults, NSKeyedArchiver & Core Data
19. **ObjC**

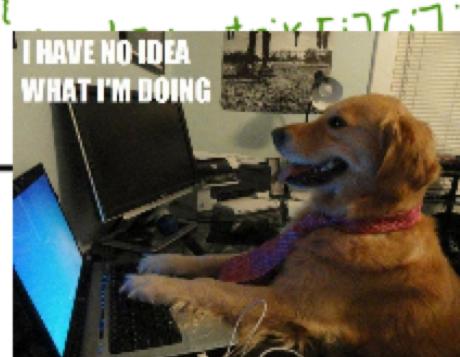
User Interfaces UIViewController, Auto Layout & Size Classes

Congratulations on making it
to the on site interview! Can
you write a program that
finds the sub matrix with the
largest sum in a N x N
matrix of integers?



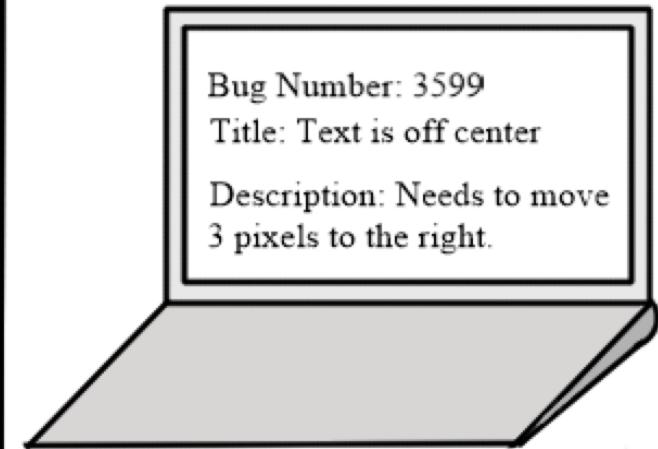
45 minutes later.

```
for(int i=0; i<N; i++)  
{ sum = 0;  
    for(int j=0; j<N; j++)  
    {
```



Day one on the job.

Bug Number: 3599
Title: Text is off center
Description: Needs to move
3 pixels to the right.



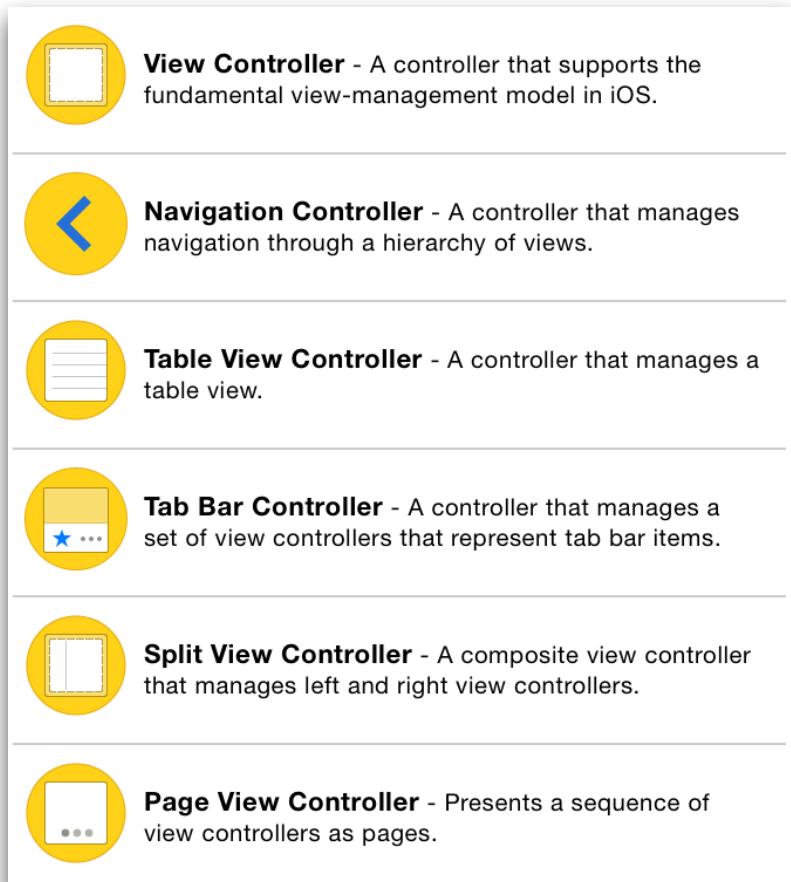
@kelukelugames

<https://twitter.com/KeLuKeLuGames/status/553404094958694400>

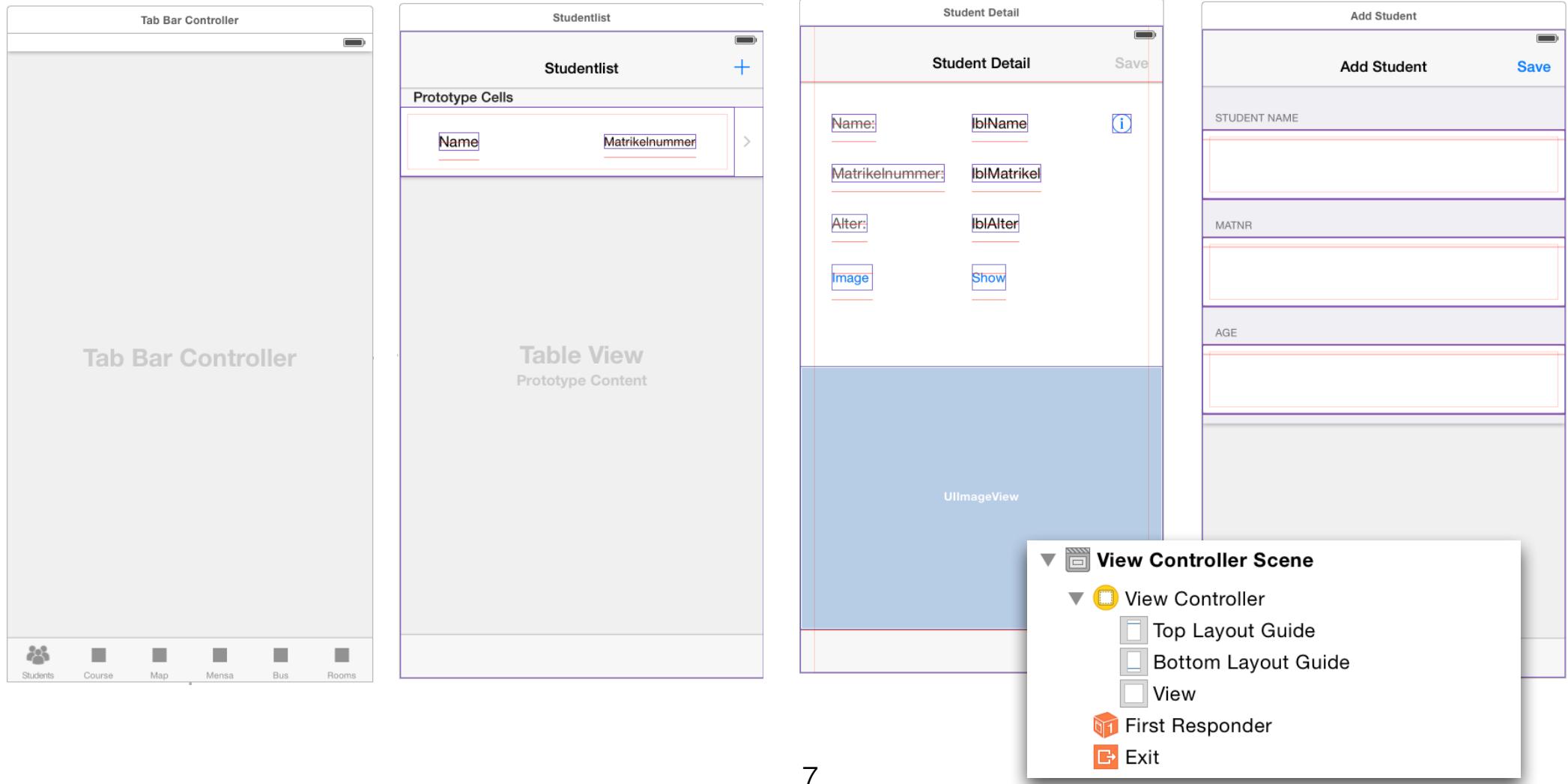
UIViewController

UIViewController

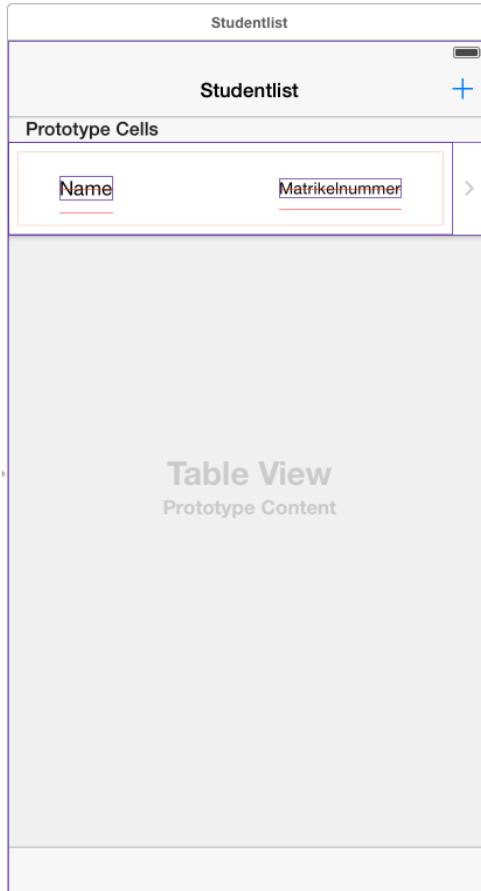
- manages one screen (typically)
 - every screen should have its own controller
- flushes view on low memory situations
- ‘resizes’ its views on orientation changes
- creates modal views on top of current view
- basically loads a ‘nib’



Examples of UIViewController



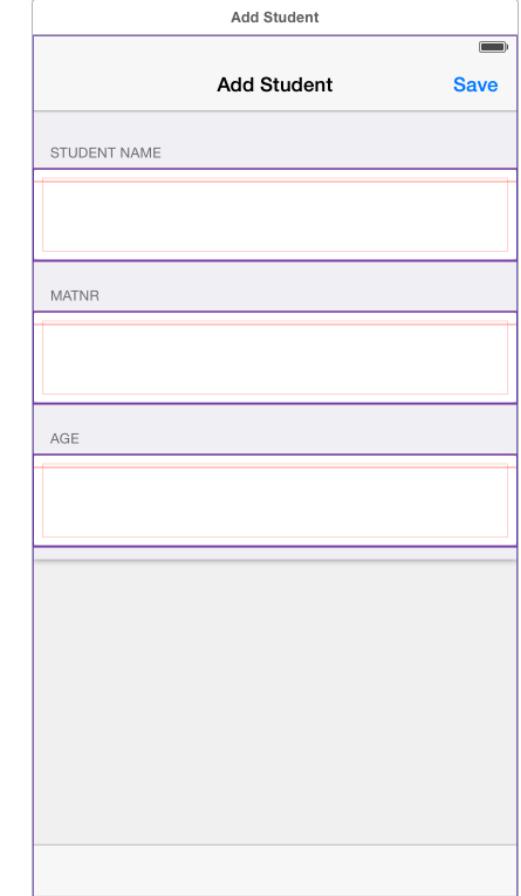
Examples of UIViewController



Custom Class:
StudentTableViewControllerBarController



Custom Class:
DetailStudentViewControllerBarController



Custom Class:
AddStudentTableViewControllerBarController

ViewController Life Cycle

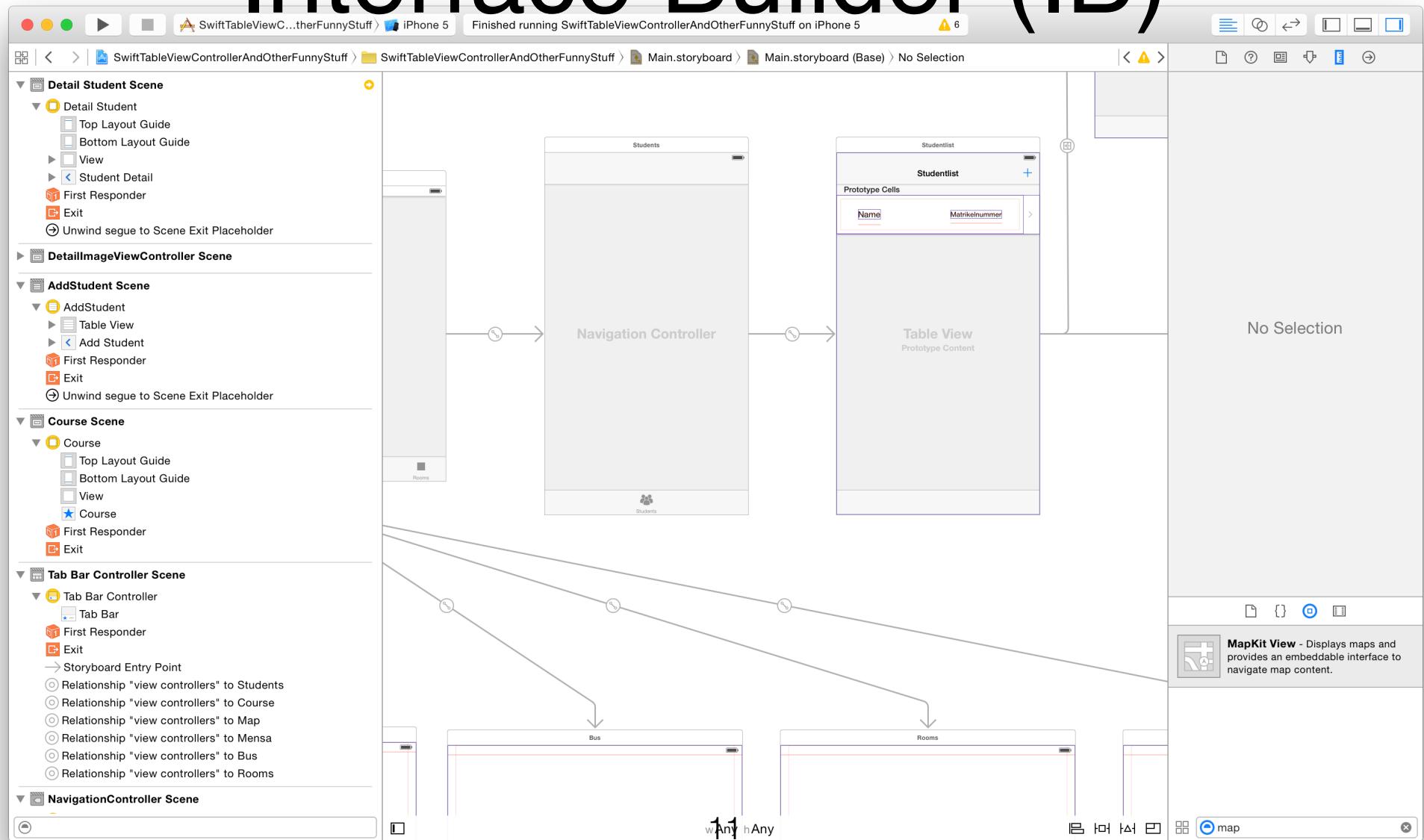
UIKit

```
init  
func loadView()  
func viewDidLoad()  
func viewDidAppear(animated: Bool)  
func viewDidDisappear(animated: Bool)  
func viewWillDidAppear(animated: Bool)  
func viewWillDisappear(animated: Bool)  
deinit()
```

User Interfaces

Auto Layout

Interface Builder (IB)



IB - Utilities (1)



Identity and Type

Name Main.storyboard
Type Default - Interface Builder Storyboard
Location Relative to Group
Base.lproj/Main.storyboard
Full Path /Users/flori/Dropbox/Talk_Swift_Uni_Würzburg_2015/Demos/SwiftTableViewCellControllerAndOtherFunnyStuff/SwiftTableViewCellControllerAndOtherFunnyStuff/Base.lproj/Main.storyboard

Interface Builder Document

Opens in Default (6.0)
Builds for Project Deployment Target (8.0)
View as iOS 7.0 and Later
 Use Auto Layout
 Use Size Classes
Global Tint Default

Localization

Base
 English Localizable Strings

Target Membership

SwiftTableViewCellControllerAndOtherFunnyStuff

Show the File inspector

Description The UILabel class implements a read-only text view. You can use this class to draw one or multiple lines of static text, such as those you might use to identify other parts of your user interface. The base UILabel class provides support for both simple and complex styling of the label text. You can also control over aspects of appearance, such as whether the label uses a shadow or draws with a highlight. If needed, you can customize the appearance of your text further by subclassing.

Availability iOS (2.0 and later)

Declared In [UILabel.h](#)

Reference [UILabel Class Reference](#)

Guides [Text Programming Guide for iOS](#)

Sample Code [AdvancedURLConnections](#), [MVCNetworking](#), [MotionGraphs](#), [MultipeerGroupChat](#), [URLCache](#)

Custom Class

Class **UILabel**
Module None

Identity

Restoration ID

User Defined Runtime Attributes

Key Path	Type	Value

Document

Label Xcode Specific Label

Object ID bwG-nZ-1N3
Lock Inherited - (Nothing)
Notes

No Font

Accessibility

Accessibility Enabled

IB - Utilities (2)



Label

Text: Plain
IbMatrikel

Color: Black

Font: System 17.0

Alignment: Left

Lines: 1

Behavior: Enabled
 Highlighted

Baseline: Align Baselines

Line Breaks: Truncate Tail

Autoshrink: Fixed Font Size
 Tighten Letter Spacing

Highlighted: Default

Shadow: Default

Shadow Offset: 0 (Horizontal) -1 (Vertical)

View: Left

Tag: 0

Interaction: User Interaction Enabled

Label

Preferred Width: Automatic
 Explicit

View: Frame Rectangle
X: 196 Y: 157
Width: 79 Height: 21

Constraints



All This Size Class

- Align Leading to: IblAlter Edit
- Align Leading to: IblName Edit
- Bottom Space to: IblAlter Equals: 36 Edit
- Top Space to: IblName Equals: 136 Edit

Outlet Collections
gestureRecognizers

Referencing Outlets
matNrLbl → * Detail Student

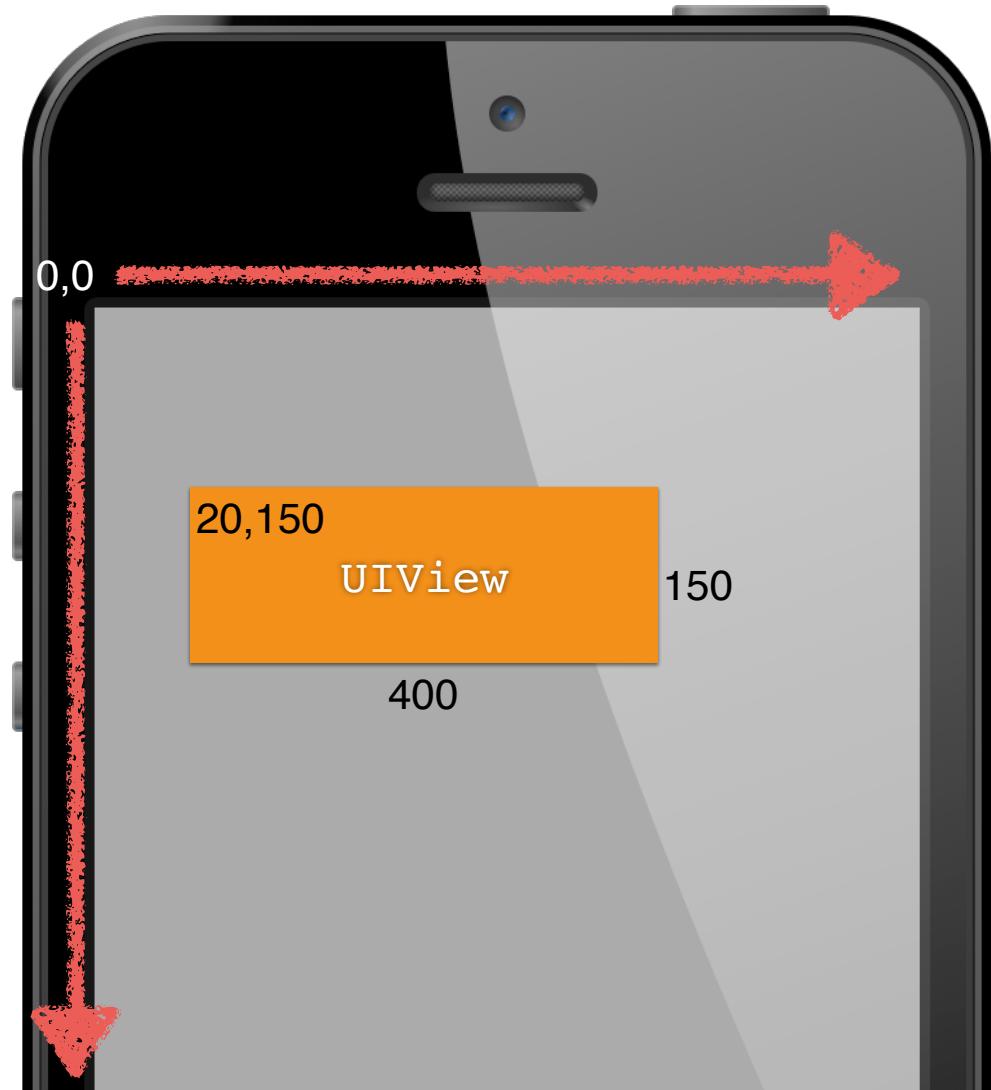
Referencing Outlet Collections
New Referencing Outlet Collection

Adding Constraints

The screenshot shows the Xcode Interface Builder environment. On the left, a storyboard scene titled "Student Detail" is displayed. It contains several UI elements: three labels ("Name:", "Matrikelnummer:", "Alter:") with their respective placeholder text fields, and two buttons labeled "Image" and "Show". A large blue `UIImageView` occupies the bottom half of the screen. On the right, the "Attributes Inspector" is open for a label, showing its frame (X: 196, Y: 157, Width: 79, Height: 21) and constraints. A modal window titled "Add New Constraints" is also visible, allowing the user to define constraints for the label. The modal shows various options like "Align Leading to: IblAlter" and "Bottom Space to: IblAlter Equals: 36". The bottom of the interface features a toolbar with icons for selection, transformation, and other tools.

Remember - UIView

- defines a rectangular area
- layout as tree
- origin top left
- Frame
 - rectangle with origin & size (relative to superview)
- Bounds
 - rectangle with origin (0,0) and size of view
- Center
 - center point of rectangle



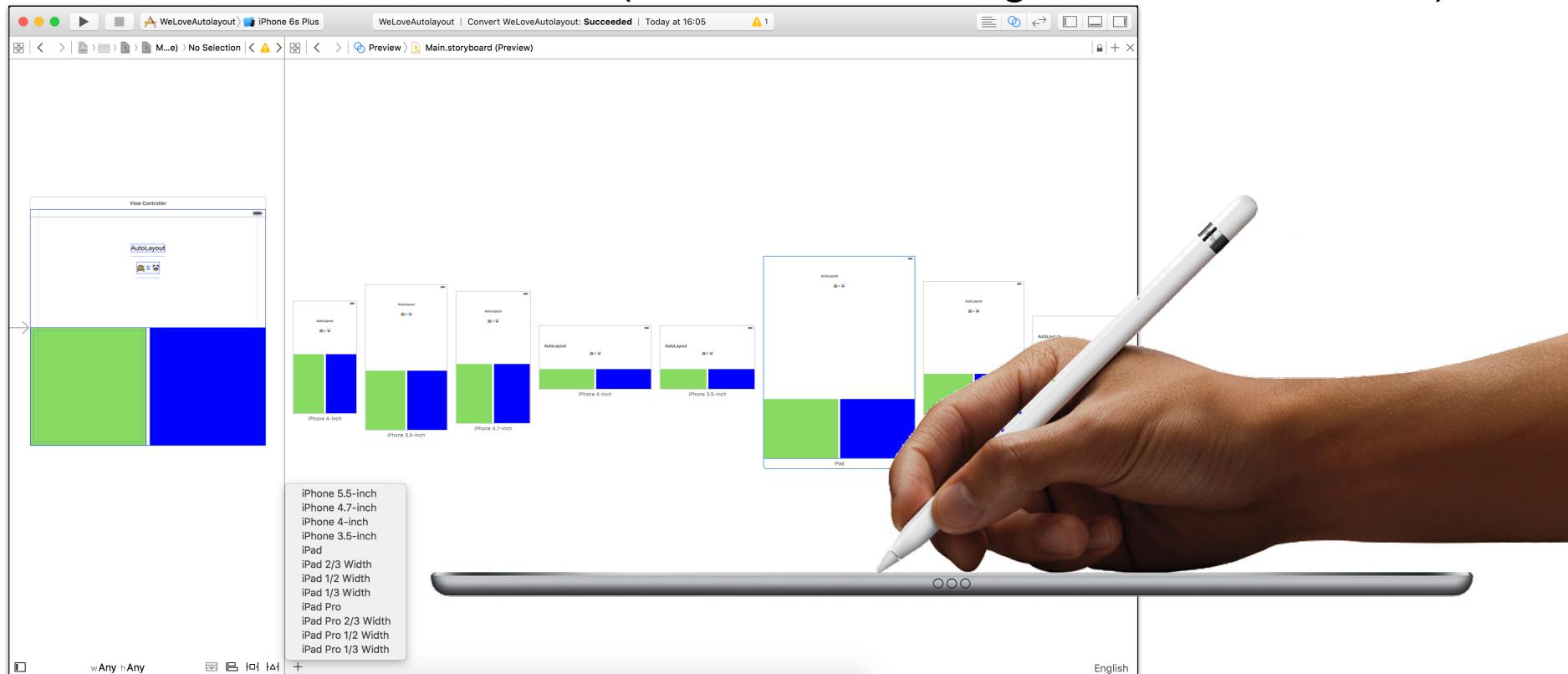
Demo Auto Layout

User Interfaces

Size Classes / Adaptive Layout

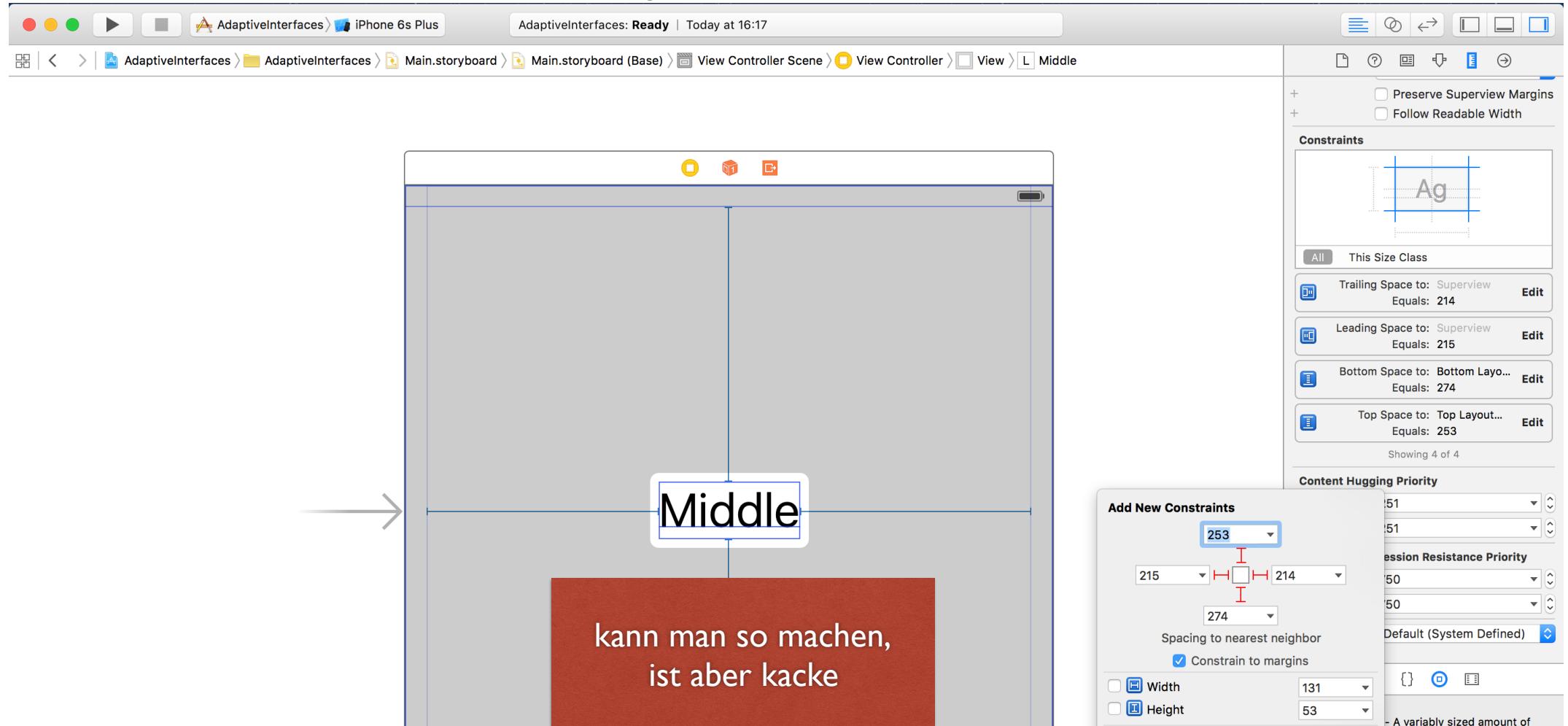
User Interface Possibilities

- Screen sizes, resolutions, and orientations
- at least 12 different form factors (immer noch weniger als bei Android :)



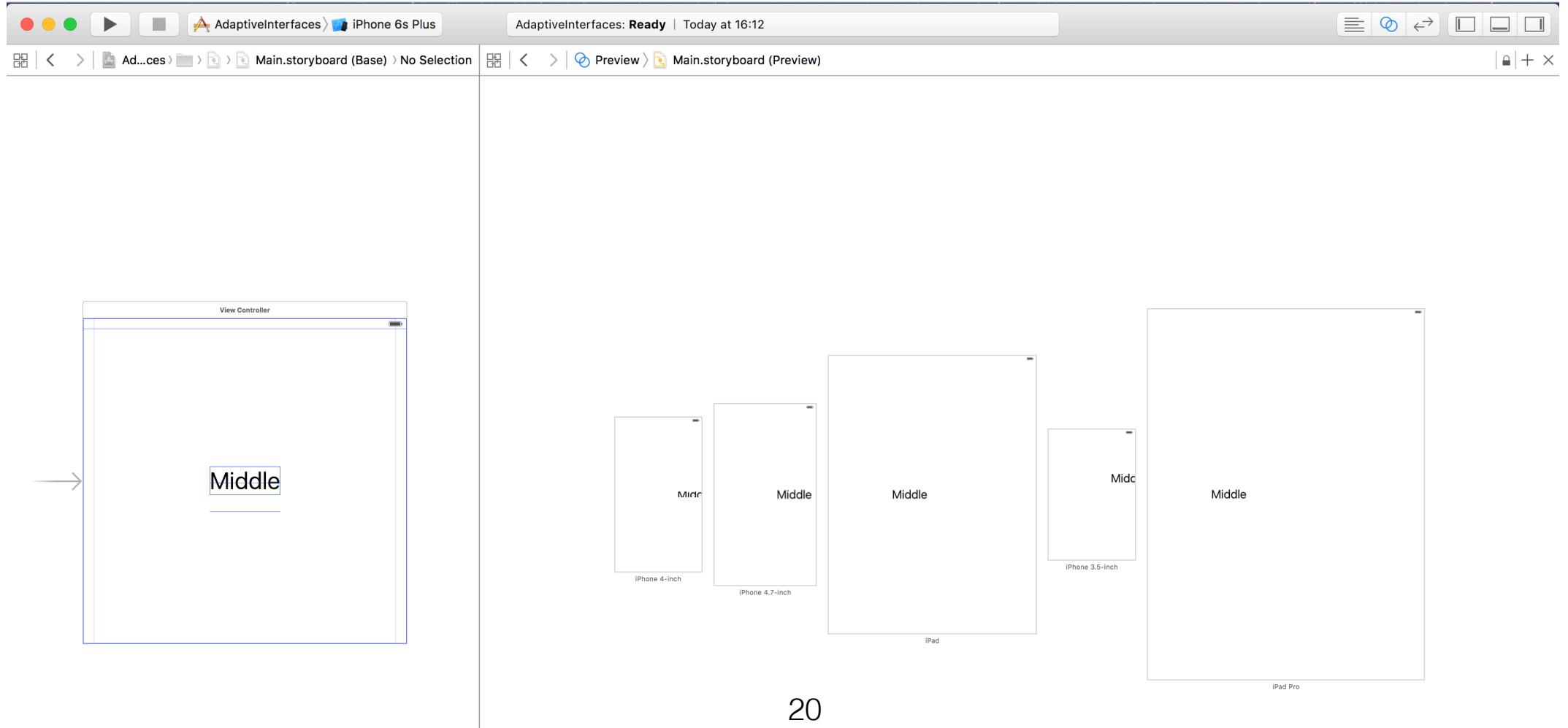
Think adaptive (1/3)

- relative position and sizing



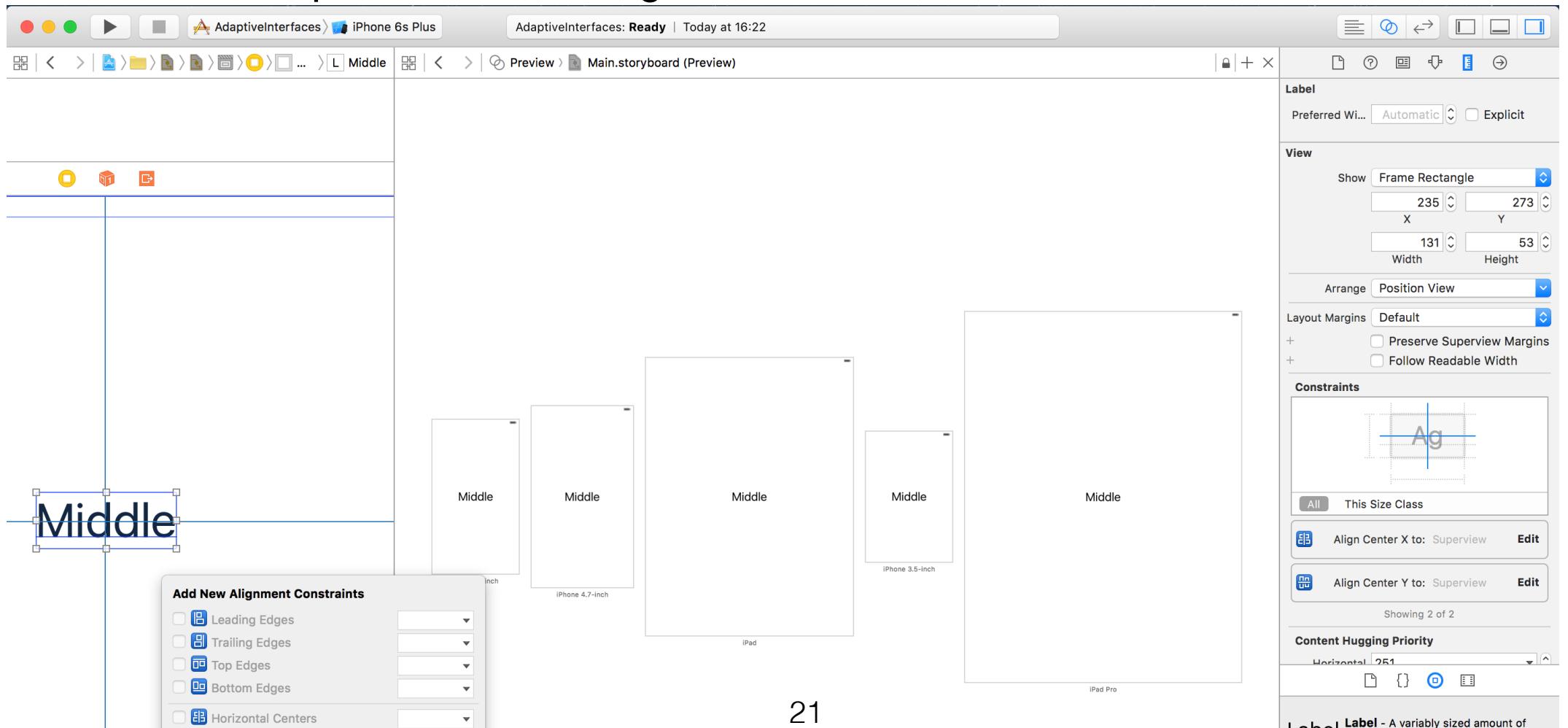
Think adaptive (2/3)

- relative position and sizing



Think adaptive (3/3)

- relative position and sizing



Adaptive Layout

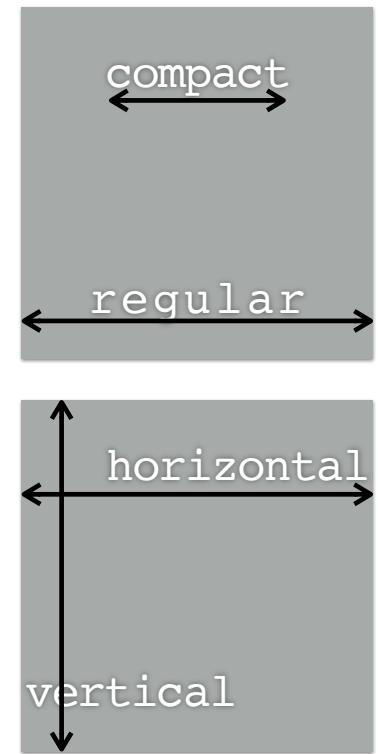
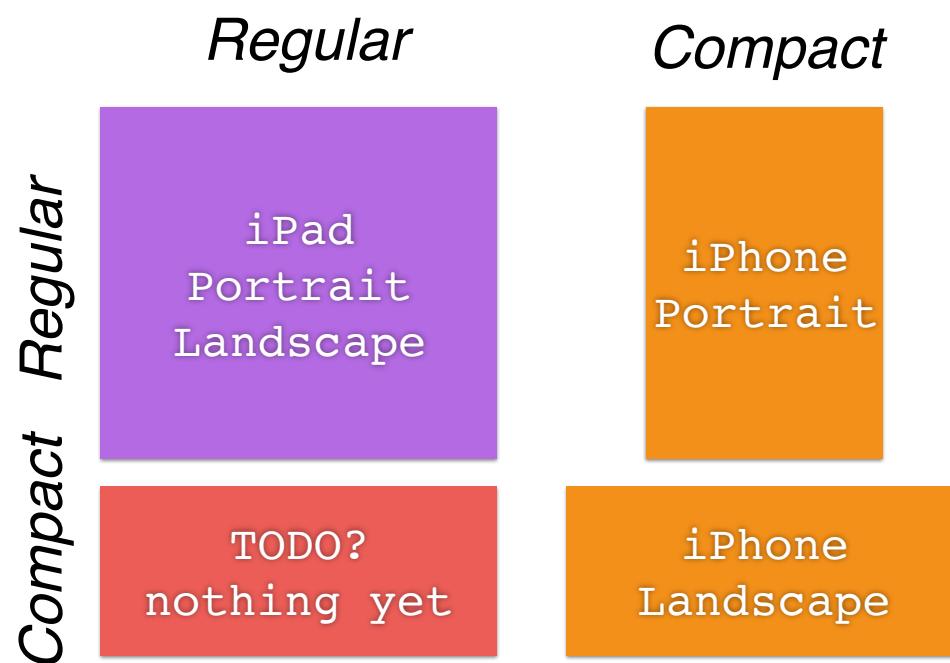
- Abstracts layout away from device specifics
- Introducing concept of size classes
- Available since iOS 8
- fully supported in IB and Xcode

Before Adaptive Layout

```
let d = UIDevice.currentDevice()
let o = d.orientation
if d.userInterfaceIdiom == .Pad {
    if o == .LandscapeLeft || o == .LandscapeRight {
        doSomethingForIpadsInLandScape()
    } else {
        yeahItIsAnIpadInPortrait()
    }
} else {
    if o == .LandscapeLeft || o == .LandscapeRight {
        itIsAnIphoneInLandscpae()
    } else {
        ohThePhoneIsInPortrait()
    }
}
```

Size Classes (1)

- *Horizontal regular, vertical regular*: iPad in either orientation
- *Horizontal compact, vertical regular*: iPhone portrait
- *Horizontal regular, vertical compact*: no current device
- *Horizontal compact, vertical compact*: iPhone landscape

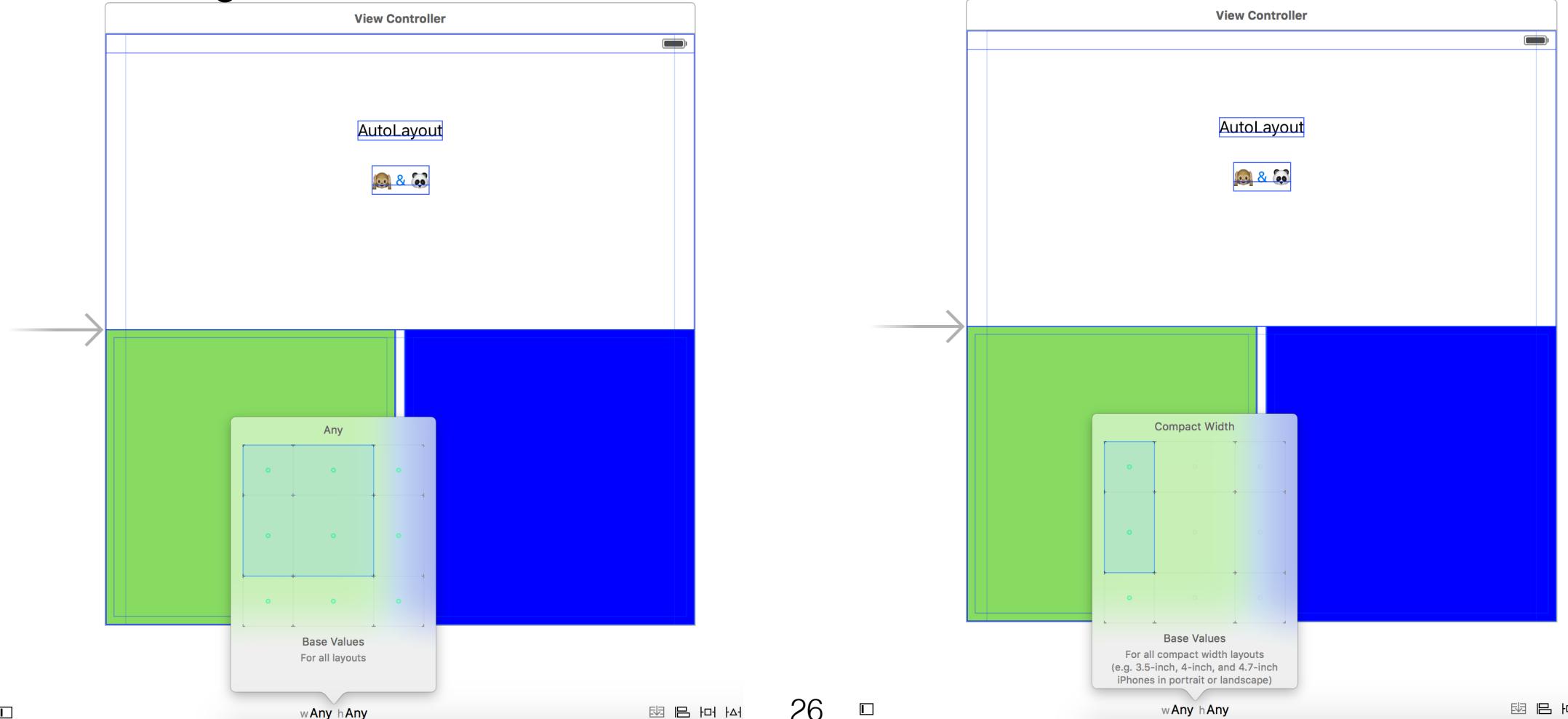


Size Classes (2)

	Vertical Size	Horizontal Size
iPad Portrait	Regular	Regular
iPad Landscape	Regular	Regular
iPhone Portrait	Regular	Compact
iPhone Landscape	Compact	Compact
iPhone 6[s] Plus Landscape	Compact	Regular

Size Classes (3)

thinking abstract about the screens



Size Classes

Specify how objects change between different size magnitudes (size classes)

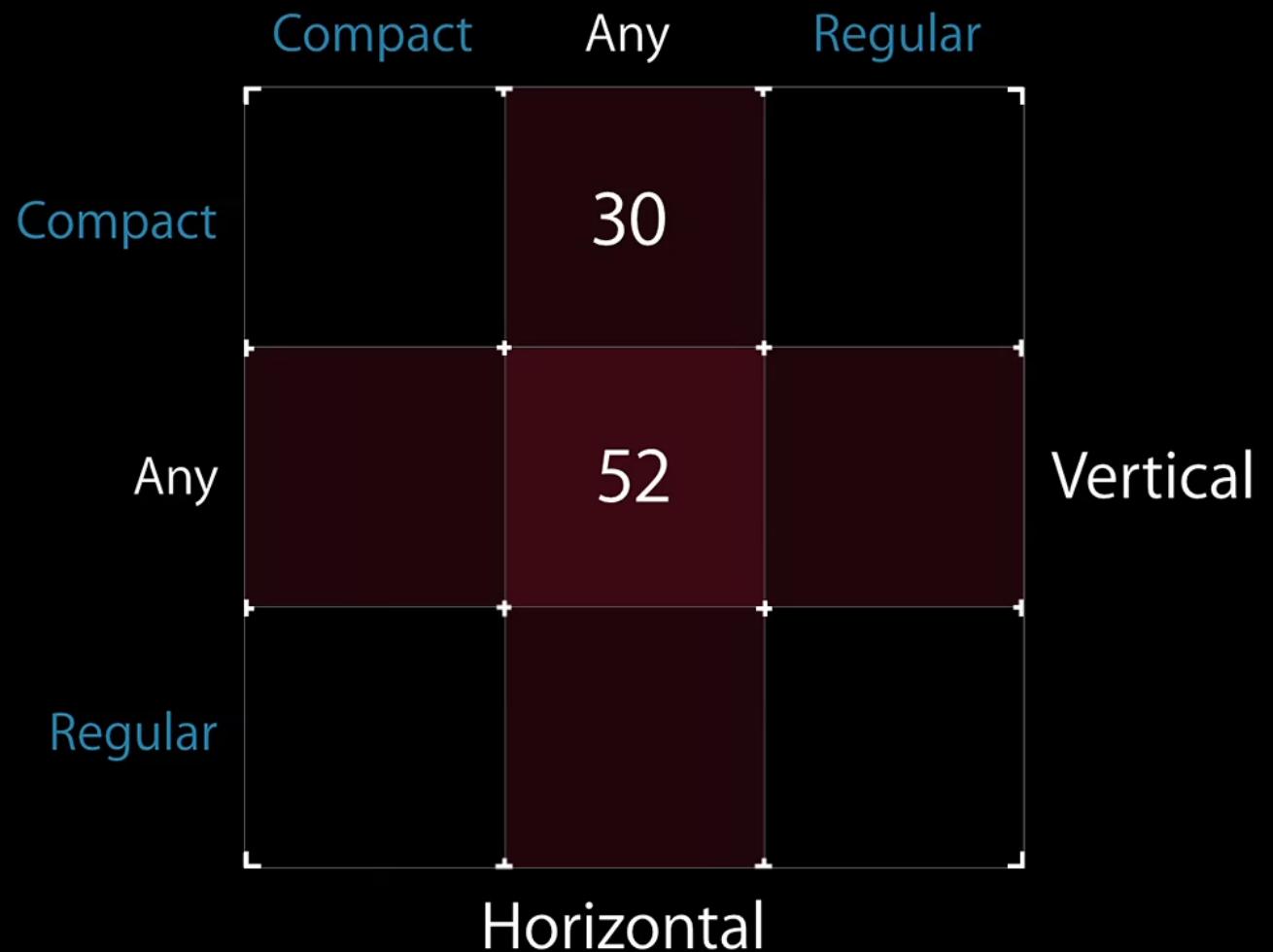
One size class per orientation

Three possible values per size class (Any, Compact, Regular)

Strategy: build in Any | Any, customize in specific configurations where needed

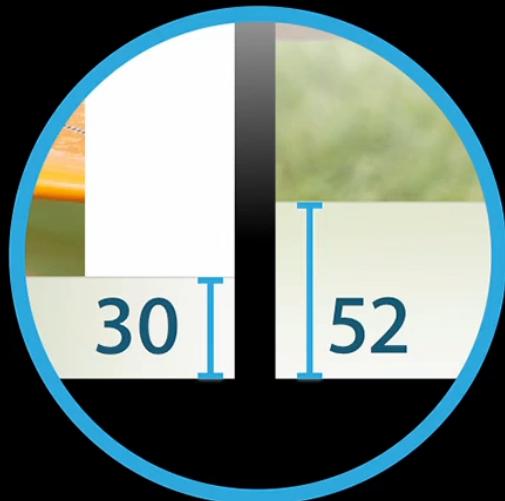
SIZE CLASSES

Example: Adapting the bar height



Size Classes

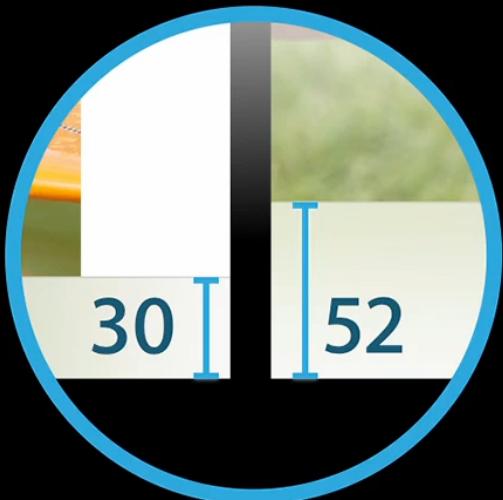
Example: Adapting the bar height



	Compact	Any	Regular
Compact		30	
Any		52	Vertical
Regular			
		Horizontal	

Size Classes

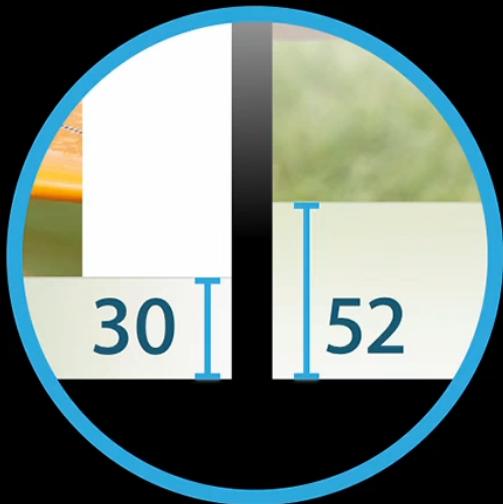
Example: Adapting the bar height



	Compact	Any	Regular
Compact		30	
Any			Vertical
Regular	52		52
	Horizontal		

Size Classes

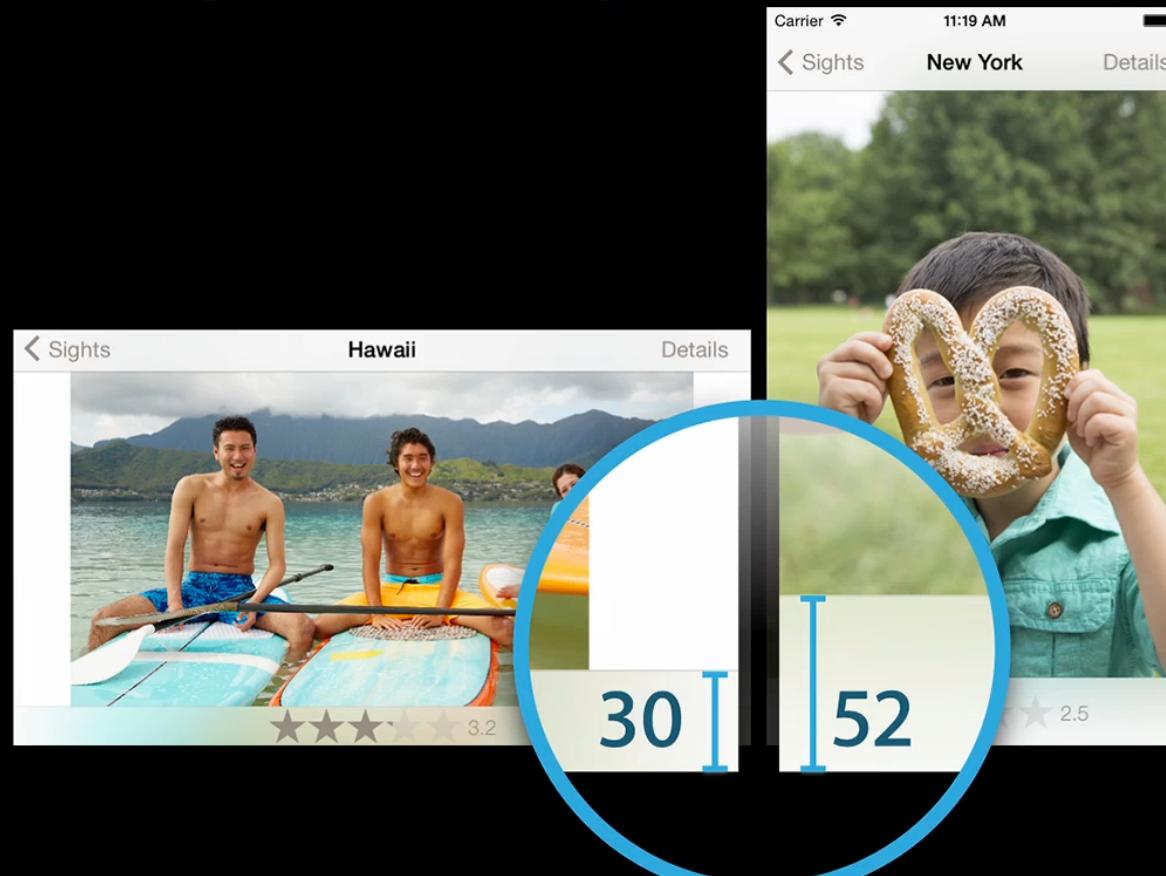
Example: Adapting the bar height



	Compact	Regular
Compact	30	30
Regular	52	52
Horizontal		
Vertical		

Size Classes

Example: Adapting the bar height



Size Classes

Search Results

Values

Compact

Regular

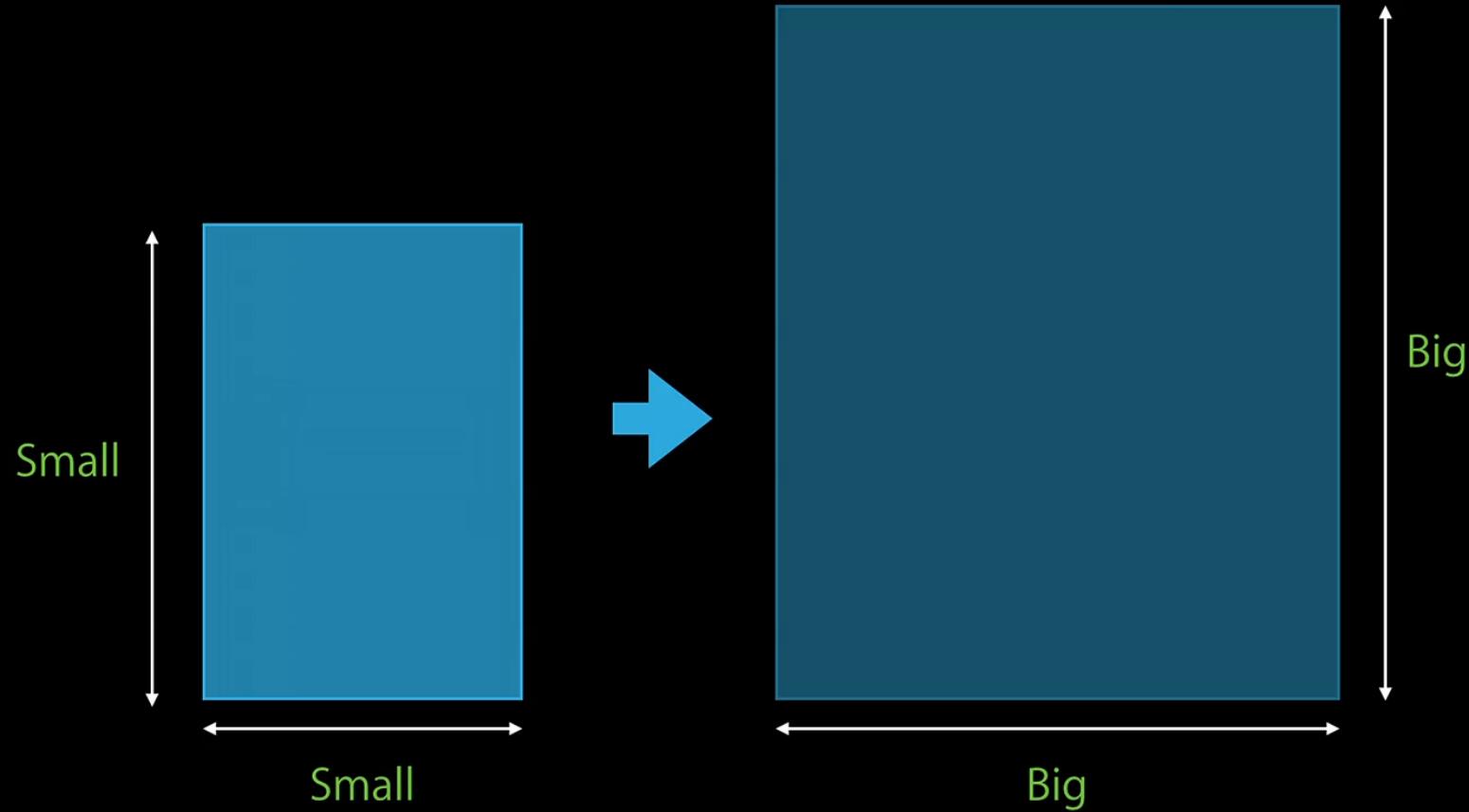
Orientation

Horizontal

Vertical

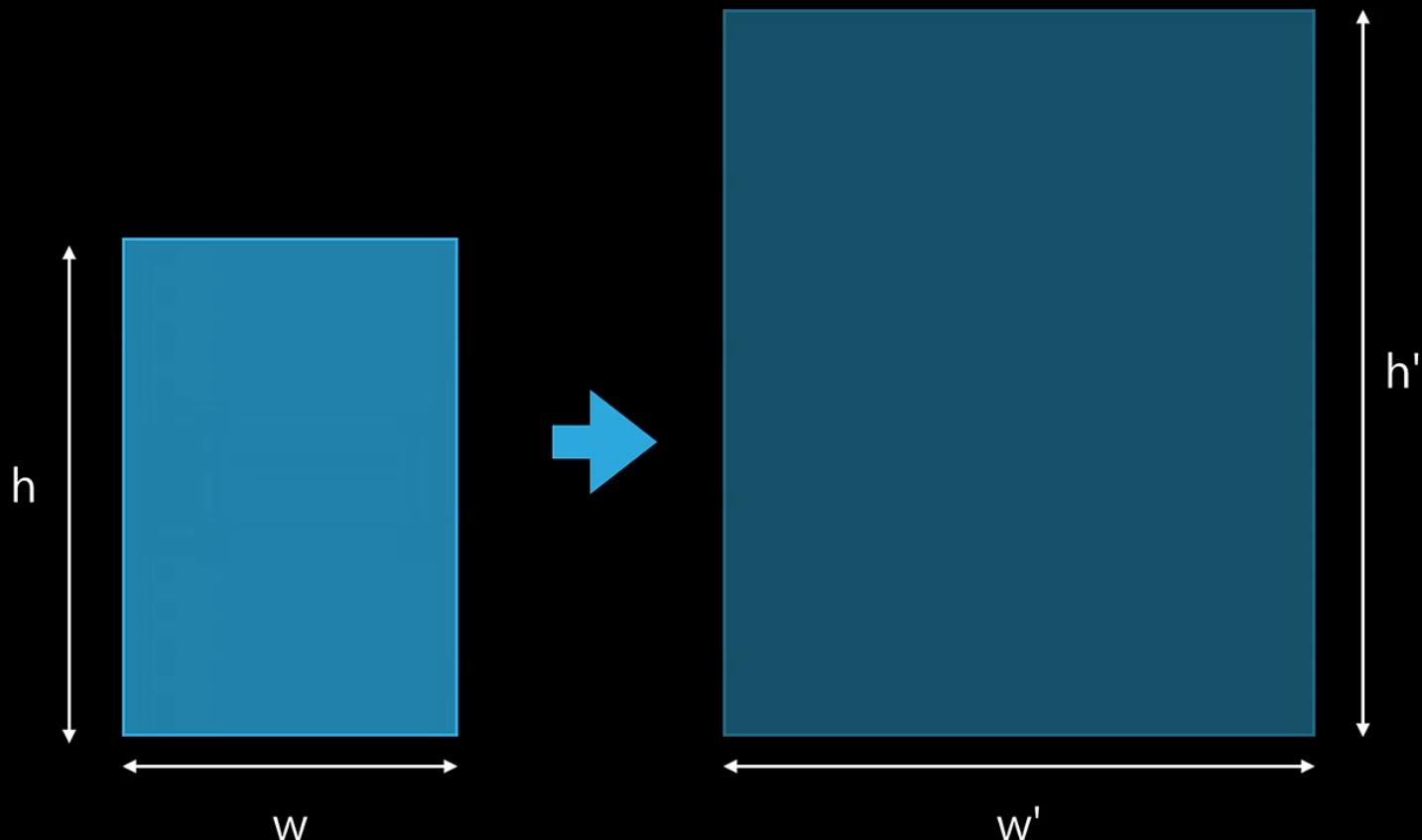
Size Classes

Thinking in terms of magnitude



Size Classes

Thinking in terms of magnitude



Allows you to specify how your UI changes when the available size of your view controller changes

**Start with
Base Layout
and then
override**

Adopting Adaptive Layout

1. *Build base layout*
2. *Choose size class override*
3. *uninstall irrelevant constraints*
4. *add new constraints specific to size class*
5. *name important constraints (with emojis ↔, ↑&✓)*
6. *repeat*

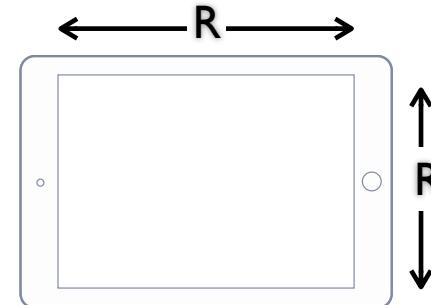
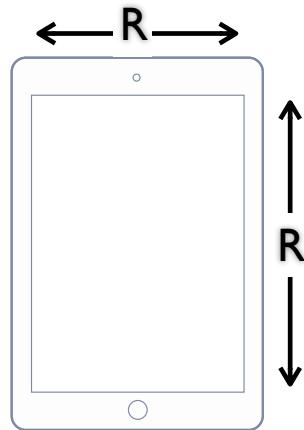
Adaptivity is:

- *constraint installation*
- *constraint constant*
- *font*
- *view installation*

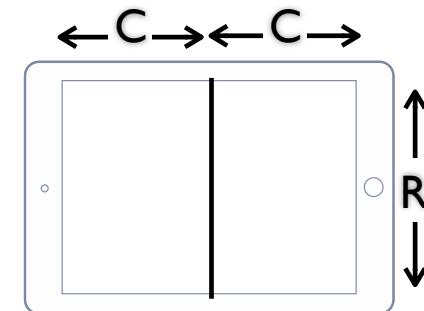
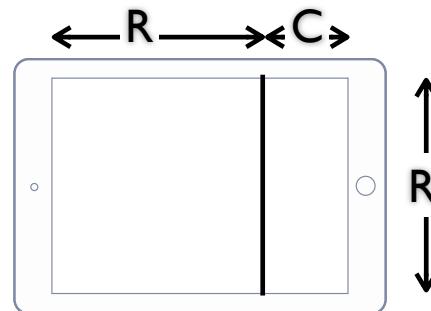
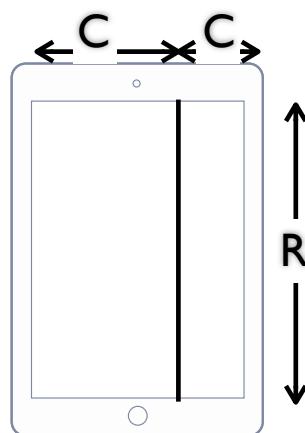
(Basically everything
with a plus symbol)

+	Font	System 17.0	
×	wAny hC	System 22.0	
×	wC hC	System 22.0	

Size Classes - Multitasking



C = compact
R = regular



UITraitEnvironments

Trait environments expose a trait collection that describes their environment.

```
public protocol UITraitEnvironment : NSObjectProtocol {  
    public var traitCollection: UITraitCollection { get }  
  
    public func traitCollectionDidChange(previousTraitCollection: UITraitCollection?)  
}
```

- UIScreen
- UIWindow
- UIPresentationController
- UIViewController
- UIView

UITraitCollection

A trait collection encapsulates the system traits of an interface's environment.

```
public protocol UITraitEnvironment : NSObjectProtocol {  
    public var traitCollection: UITraitCollection { get }  
  
    public func traitCollectionDidChange(previousTraitCollection: UITraitCollection?)  
}
```

- UIScreen
- UIWindow
- UIPresentationController
- UIViewController
- UIView

UIContentContainer

React to changes in Code

```
public protocol UIContentContainer : NSObjectProtocol {  
    ... systemLayoutFittingSizeDidChangeForChildContentContainer(  
        container: UIContentContainer  
    )  
  
    ... viewWillTransitionToSize(  
        size: CGSize,  
        withTransitionCoordinator coordinator: UIViewControllerTransitionCoordinator  
    )  
  
    ... willTransitionToTraitCollection(  
        newCollection: UITraitCollection,  
        withTransitionCoordinator coordinator: UIViewControllerTransitionCoordinator  
    )
```

UIContentContainer Rotation - Example

...

```
override func viewWillTransitionToSize(  
    size: CGSize,  
    withTransitionCoordinator coordinator: UIViewControllerTransitionCoordinator  
) {  
    super.viewWillTransitionToSize(size, withTransitionCoordinator: coordinator)  
    let image = imageForAspectRatio(size.width / size.height)  
  
    coordinator.animateAlongsideTransition({  
        context in  
        let transition = CATransition()  
        transition.duration = context.transitionDuration()  
        transition.timingFunction = CAMediaTimingFunction(...: .EaseInEaseOut)  
        self.backgroundImageView.layer.addAnimation(transition.duration, forKey: "Fade")  
        self.backgroundImageView.image = image  
    }, completion: nil)
```

Demo

Size Classes / Adaptive Layout

Errors & Debugging

Autolayout Errors - WTF?

Unable to simultaneously satisfy constraints.

Probably at least one of the constraints in the following list is one you don't want.

Try this:

- (1) look at each constraint and try to figure out which you don't expect;
- (2) find the code that added the unwanted constraint or constraints and fix it.

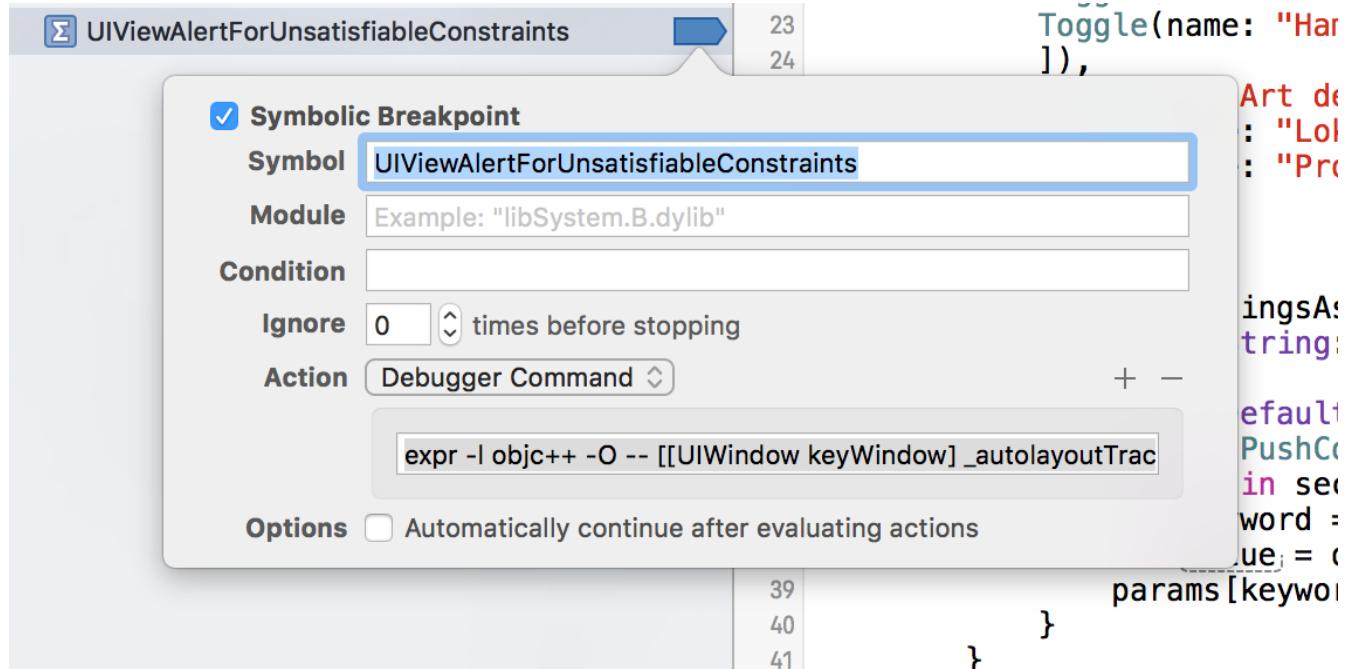
(Note: If you're seeing `NSAutoresizingMaskLayoutConstraint`s that you don't understand, refer to the documentation for the `UIView` property `translatesAutoresizingMaskIntoConstraints`)

(.....)

Make a symbolic breakpoint at `UIViewAlertForUnsatisfiableConstraints` to catch this in the debugger.

The methods in the `UIConstraintBasedLayoutDebugging` category on `UIView` listed in `<UIKit/UIKit.h>` may also be helpful.

Auto Layout Breakpoints - FTW!



Symbol:
UIAlertViewForUnsatisfiableConstraints

Action:
expr -l objc++ -O -- [[UIWindow keyWindow] _autolayoutTrace]

Auto Layout Breakpoints

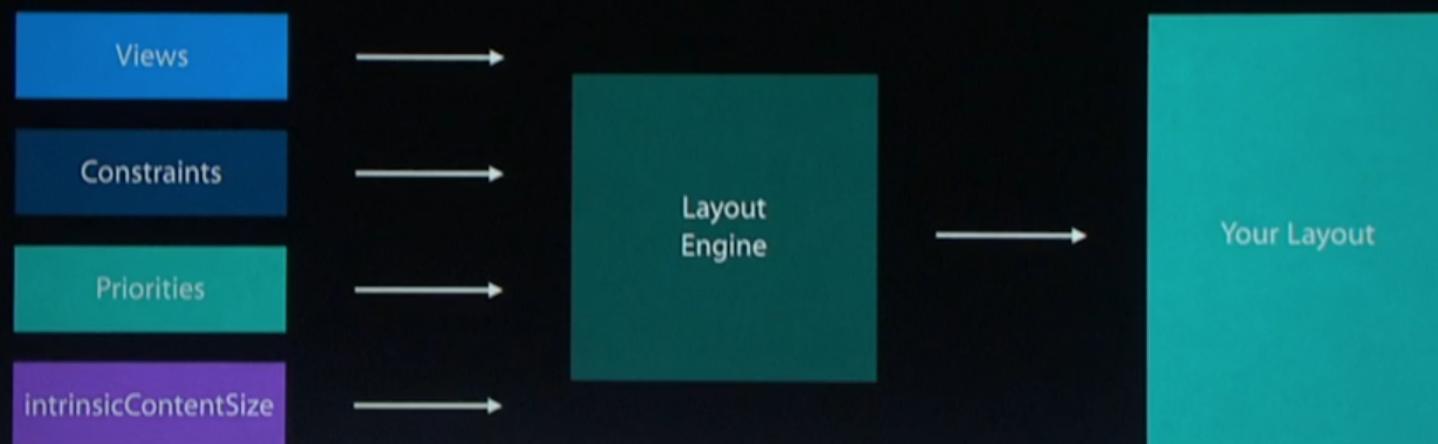
- gibt uns -> gute Konsolen-Ausgaben

```
UIWindow:0x7f9481c93360
|   •UIView:0x7f9481c9d680
|   |   *UIView:0x7f9481c9d990- AMBIGUOUS LAYOUT for UIView:0x7f9481c9d990.minX{id: 13}, UIView:
0x7f9481c9d990.minY{id: 16}
|   |   *_UILayoutGuide:0x7f9481c9e160- AMBIGUOUS LAYOUT for _UILayoutGuide:0x7f9481c9e160.minY{id: 17}
|   |   *_UILayoutGuide:0x7f9481c9ebb0- AMBIGUOUS LAYOUT for _UILayoutGuide:0x7f9481c9ebb0.minY{id: 27}
```

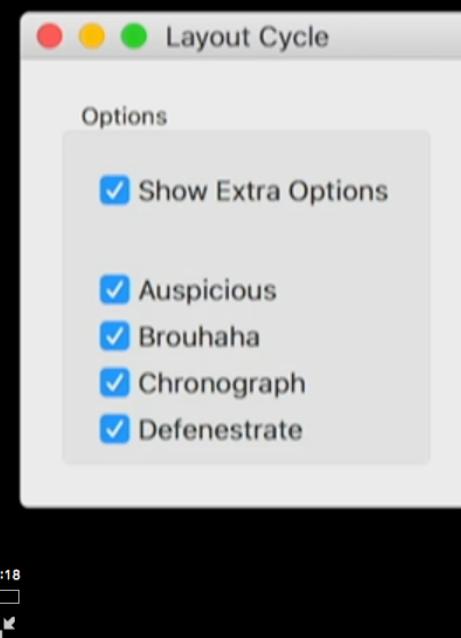
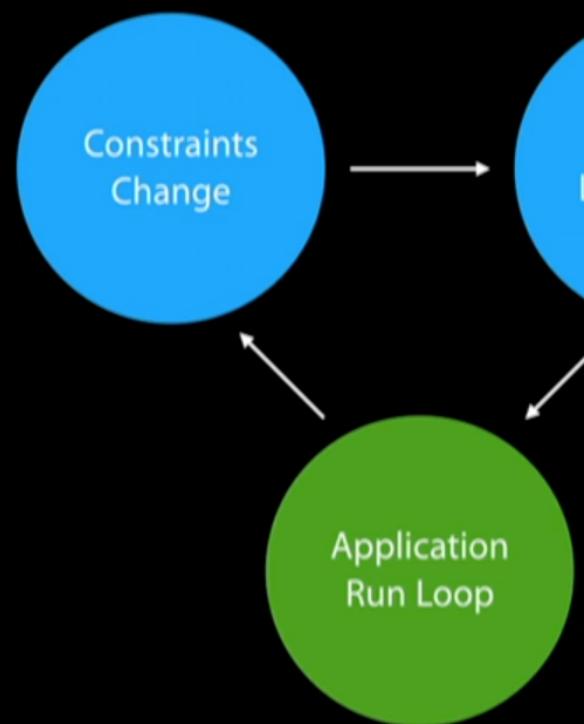
- die wir dennoch nicht wirklich verstehen 🤔
- die Lösung: Hintergrundfarbe des Übeltäters ändern 😎

```
(lldb) expr ((UIView *)0x7f9ea3d43410).backgroundColor = [UIColor redColor]
(UICachedDeviceRGBColor *) $1 = 0x00007f9ea3d43410
```

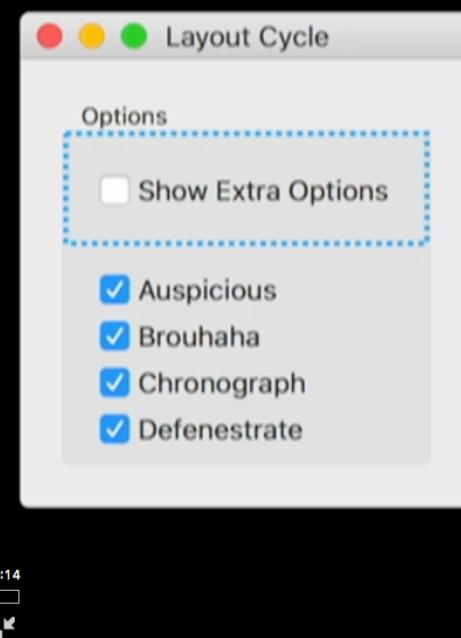
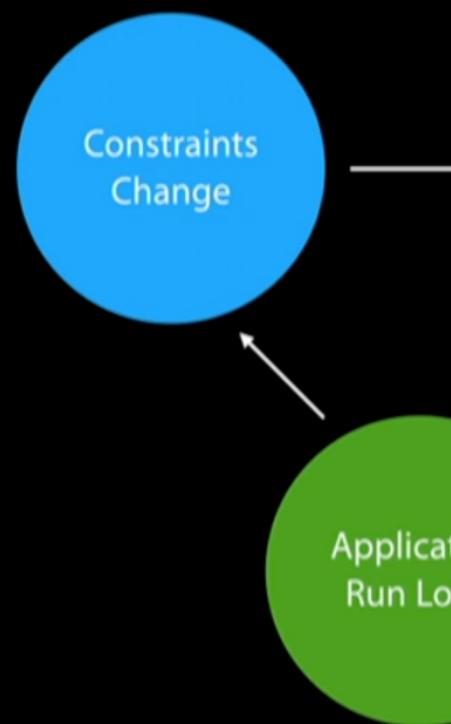
Inside the Black Box



The Layout Cycle



The Layout Cycle



Resolving Ambiguity

Diagnostic tools

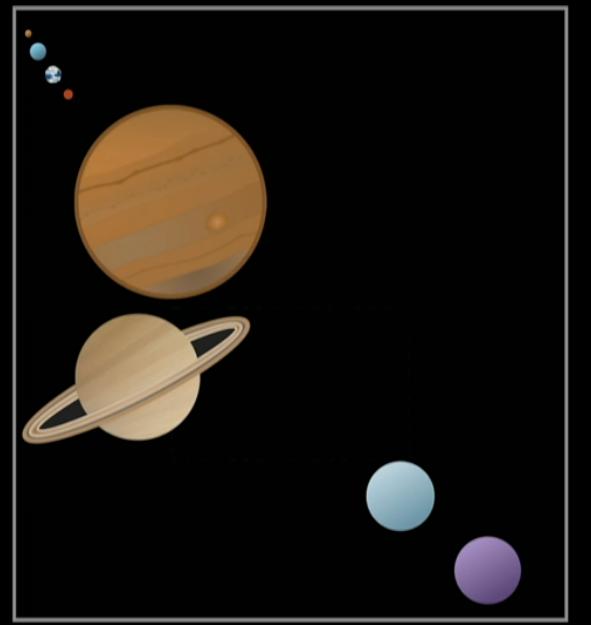
Red and yellow icons in IB

`_autoLayoutTrace`

Select Debug > View Debugging

Look in the view debugger

`exerciseAmbiguityInLayout`



Resolving Ambiguity

Diagnostic tools

Red and yellow icons in IB

`_autolayoutTrace`

Select Debug > View Debugging

Look in the view debugger



Resolving Ambiguity

Diagnostic tools

Red and yellow icons in IB

`_autolayoutTrace`

Select Debug > View Debugging

Look in the view debugger



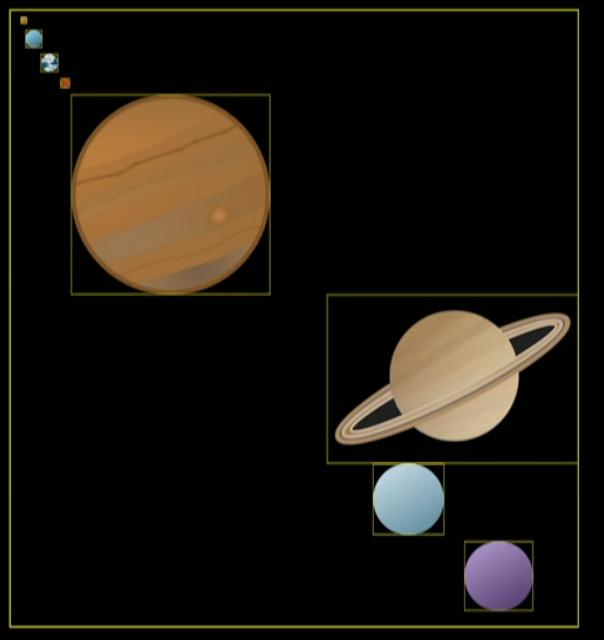
Resolving Ambiguity

Diagnostic tools

Red and yellow icons in IB

`_autolayoutTrace`

Select Debug > View Debugging



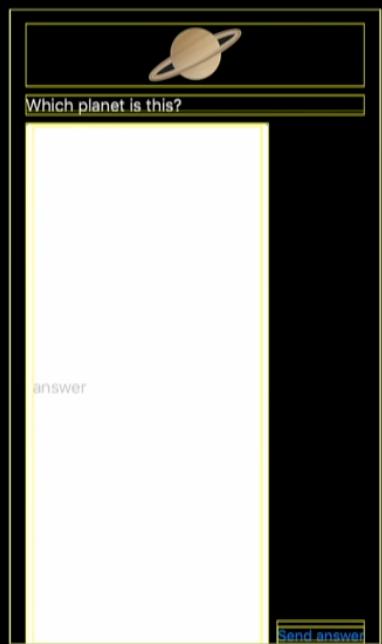
Resolving Ambiguity

Diagnostic tools

Red and yellow icons in IB

`_autoLayoutTrace`

Select Debug > View Debugging

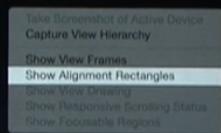


Solving Ambiguity Diagnostic tools

red and yellow icons in IB

`autoLayoutTrace`

Select Debug > View Debugging



Resolving Ambiguity

Diagnostic tools

Red and yellow icons in IB

_autolayoutTrace

```
(lldb) po [self.view _autolayoutTrace]
UIWindow:0x7fe7434a3fe0
| *UIView:0x7fe7434a8140
| | *_UILayoutGuide:0x7fe7434a84f0
| | *_UILayoutGuide:0x7fe7434a90d0
| | *Mercury:0x7fe7434a7790
| | *Venus:0x7fe743639380
| | *Earth:0x7fe74363aae0
| | *Mars:0x7fe74363bed0
| | *Jupiter:0x7fe74363ce30
| | *Saturn:0x7fe74363e220- AMBIGUOUS LAYOUT for Saturn.minX{id: 165}
| | *Uranus:0x7fe74363f690
| | *Neptune:0x7fe743640d60

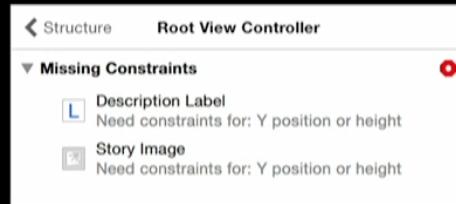
Legend:
* - is laid out with auto layout
+ - is laid out manually, but is represented in the layout engine
because translatesAutoresizingMaskIntoConstraints = YES
• - layout engine host

(lldb)
```

Resolving Ambiguity

Diagnostic tools

Red and yellow icons in IB

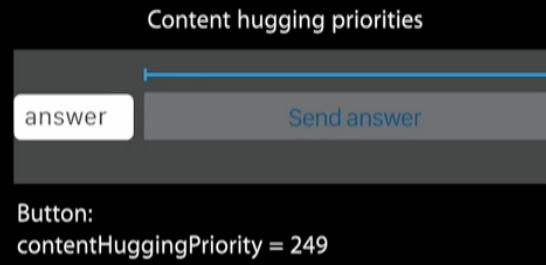


Ambiguous Layouts

Why doesn't my layout look right?

Possible causes

- Too few constraints
- Conflicting priorities



Understanding the Log

Start from the bottom

Check `translatesAutoresizingMaskIntoConstraints`

Set identifiers

Use `constraintsAffectingLayoutForAxis:`

Understanding the Log

Adding identifiers

Use constraint identifiers

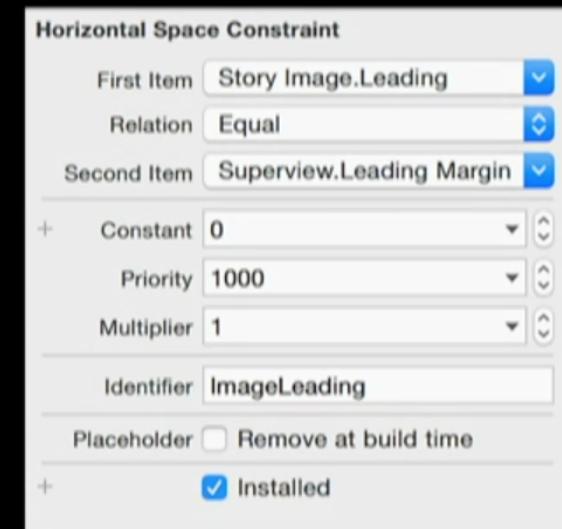
Explicit constraints

```
labelToTop.identifier = @"labelToTop";
```

Constraints using VFL

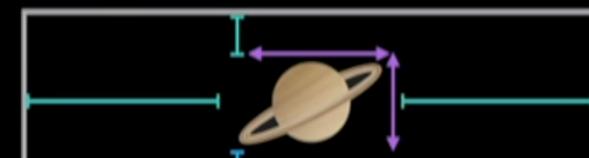
```
for (NSLayoutConstraint *constraint in verticalLayout)
{
    constraint.identifier = @"verticalLayout";
}
```

Constraints in Interface Builder



Understanding the Log

```
(  
    "<_UILayoutSupportConstraint:0x7ffe9ad11a80 V:[_UILayoutGuide:  
0x7ffe9ad10650(20)]>",  
    "<_UILayoutSupportConstraint:0x7ffe9ad10ba0 V:|-(0)-[_UILayoutGuide:  
0x7ffe9ad10650] (Names: '|':UIView:0x7ffe9c81b720 )>",  
    "<NSLayoutConstraint:0x7ffe9acbef60 'saturnWidth' saturn.width ==  
1.5*saturn.height (Names: saturn:0x7ffe9acb8cb0 )>",  
    "<NSLayoutConstraint:0x7ffe9c905460 'imageHorizontal' saturn.leading  
== UIView:0x7ffe9c81b720.leadingMargin (Names: saturn:  
0x7ffe9acb8cb0 )>",  
    "<NSLayoutConstraint:0x7fedd3423ae0 'imageHorizontal' UIView:  
0x7fedd3607b90.trailingMargin == saturn.trailing>, (Names: saturn:  
0x7ffe9acb8cb0 )>",  
    "<NSLayoutConstraint:0x7ffe9c905aa0 'verticalLayout' V:  
[_UILayoutGuide:0x7ffe9ad10650]-(NSSpace(8))-[saturn] (Names: saturn:  
0x7ffe9acb8cb0 )>",  
    "<NSLayoutConstraint:0x7ffe9c905b40 'verticalLayout' V:[saturn]-  
(NSSpace(8))-[UILabel:0x7ffe9c903d10'Which planet is this?'] (Names:  
saturn:0x7ffe9acb8cb0 )>",  
    "<NSLayoutConstraint:0x7ffe9c906050 'labelToTop' V:|-(100)-[UILabel:  
0x7ffe9c903d10'Which planet is this?'] (Names: '|':UIView:0x7ffe9c81b720  
)>",  
    "<NSLayoutConstraint:0x7ffe9aca0130 'UIView-Encapsulated-Layout-Width'  
H:[UIView:0x7ffe9c81b720(375)]>"  
)  
  
Will attempt to recover by breaking constraint  
<NSLayoutConstraint:0x7ffe9acbef60 'saturnWidth' saturn.width ==  
1.5*saturn.height (Names: saturn:0x7ffe9acb8cb0 )>
```



Which planet is this?

answer

Send answer

A diagram of the planet Saturn is shown, featuring its characteristic rings. Two green double-headed arrows are drawn around the planet: one horizontal arrow indicates the width of the planet, and a vertical arrow indicates its height. Below the diagram, the text "Which planet is this?" is displayed. A large white rectangular box is positioned below the question, intended for user input. At the bottom right of this box, there is a button labeled "Send answer".

NSLayoutGuide / UILayoutGuide

NEW

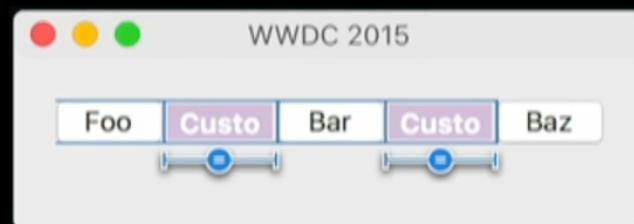
UILayoutGuide represents a rectangle in the layout engine

Constrain just like a view

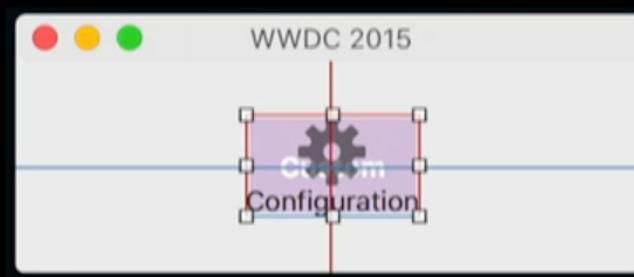
```
let guide = UILayoutGuide()  
view.addLayoutGuide(guide)
```

Constraining Negative Space

Equal spacing between buttons



Centering a group



Layout Constraint Creation

NEW

Layout anchors

```
NSLayoutConstraint(item:b, attribute:.Top, relatedBy:.Equal, toItem:view,  
attribute:.Top, multiplier:1, constant:10)
```

```
NSLayoutConstraint(item:b, attribute:.Leading, relatedBy:.Equal,  
toItem:view, attribute:.Leading, multiplier:1, constant:10)
```

```
b.topAnchor.constraintEqualToAnchor(view.topAnchor, constant:10)
```

```
b.leadingAnchor.constraintEqualToAnchor(view.leadingAnchor, constant:10)
```

Layout Constraint Creation

```
NSLayoutConstraint(item:b, attribute:.Top, relatedBy:.Equal, toItem:view,  
    attribute:.Top, multiplier:1, constant:10)  
NSLayoutConstraint(item:b, attribute:.Leading, relatedBy:.Equal,  
    toItem:view, attribute:.Leading, multiplier:1, constant:10)
```

translatesAutoresizingMaskIntoConstraints

Do not forget to turn this off!

```
override func viewDidLoad() {
    super.viewDidLoad()

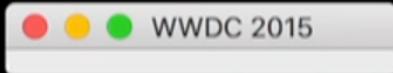
    let b = NSButton()
    b.bezelStyle = .RoundedBezelStyle
b.translatesAutoresizingMaskIntoConstraints = false
    view.addSubview(b)

    NSLayoutConstraint(item:b, attribute:.Top, relatedBy:.Equal, toItem:view,
        attribute:.Top, multiplier:1, constant:10).active = true
    NSLayoutConstraint(item:b, attribute:.Leading, relatedBy:.Equal,
        toItem:view, attribute:.Leading, multiplier:1, constant:
        10).active = true
}
```

translatesAutoresizingMaskIntoConstraints

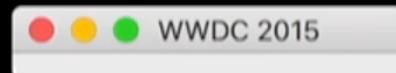
Do not forget to turn this off!

```
2015-05-08 09:41:27.668 WWDC 2015[4107:226949] Unable to simultaneously
satisfy constraints:
(
    "<NSAutoresizingMaskLayoutConstraint:0x6100000810e0 h=---& v=---& H:|-(>)-[NSButton:0x618000140160'Button']  (Names: '|':NSView:0x618000120460 )>",
    "<NSLayoutConstraint:0x6180000828a0 H:|-(10)-[NSButton:
0x618000140160'Button'](LTR)  (Names: '|':NSView:0x618000120460 )>"
)
Will attempt to recover by breaking constraint
<NSLayoutConstraint:0x6000000825d0 H:|-(10)-[NSButton:0x600000140c60'Button']
(LTR)  (Names: '|':NSView:0x6000001203c0 )>
```



`translatesAutoresizingMaskIntoConstraints`

Do not forget to turn this off!



translatesAutoresizingMaskIntoConstraints

Do not forget to turn this off!

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    let b = NSButton()  
    b.bezelStyle = .RoundedBezelStyle  
  
    view.addSubview(b)  
  
    NSLayoutConstraint(item:b, attribute:.Top, relatedBy:.Equal, toItem:view,  
        attribute:.Top, multiplier:1, constant:10).active = true  
    NSLayoutConstraint(item:b, attribute:.Leading, relatedBy:.Equal,  
        toItem:view, attribute:.Leading, multiplier:1, constant:  
        10).active = true  
}
```



`translatesAutoresizingMaskIntoConstraints`

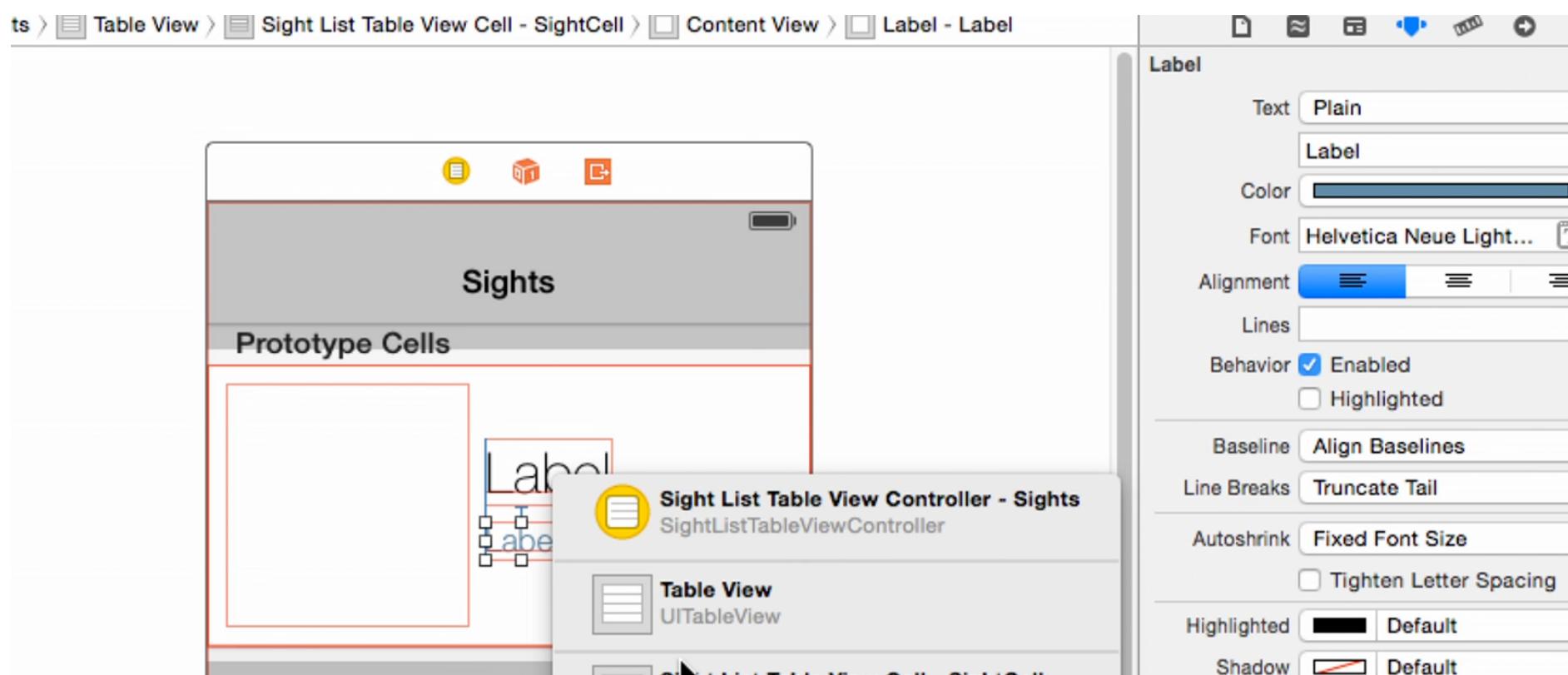
Setting the frame automatically generates constraints

- Set the frame with gleeful abandon!
- Constraints implement the autoresizingMask
- Other views can be constrained to it

Set to false when using constraints

- Beware—defaults to true for programmatically created views

Tipp: Shift



Look out for UILayoutGuide*

*Unfortunately Code only at the moment

UILayoutGuide

IMPORTANT

This is a preliminary document for an API or technology in development. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein for use on Apple-branded products. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future betas of the API or technology.

Inherits From



Conforms To

`CVarArgType`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSCoding`
`NSObjectProtocol`

Import Statement

`SWIFT`
`import UIKit`

Availability

Available in iOS 9.0 and later

The `UILayoutGuide` class defines a rectangular area that can interact with Auto Layout.

“Really learn Interface Builder
This makes prototyping much faster”

Chris Eidhof